Fatec Praia Grande

Paulo R. T. Cândido

NodeJS - parte 4

Controle de acesso

1) Instalar pacote pra criptografia

yarn add bcryptjs

yarn add @types/bcryptjs -D

2) Instalar JWT (Json Web Token), gereador de tokens de segurança

yarn add jsonwebtoken

yarn add @types/jsonwebtoken -D

3) Instalat pacote para acessar variáveis em .env

yarn add dotenv

4) Criar model Usuario

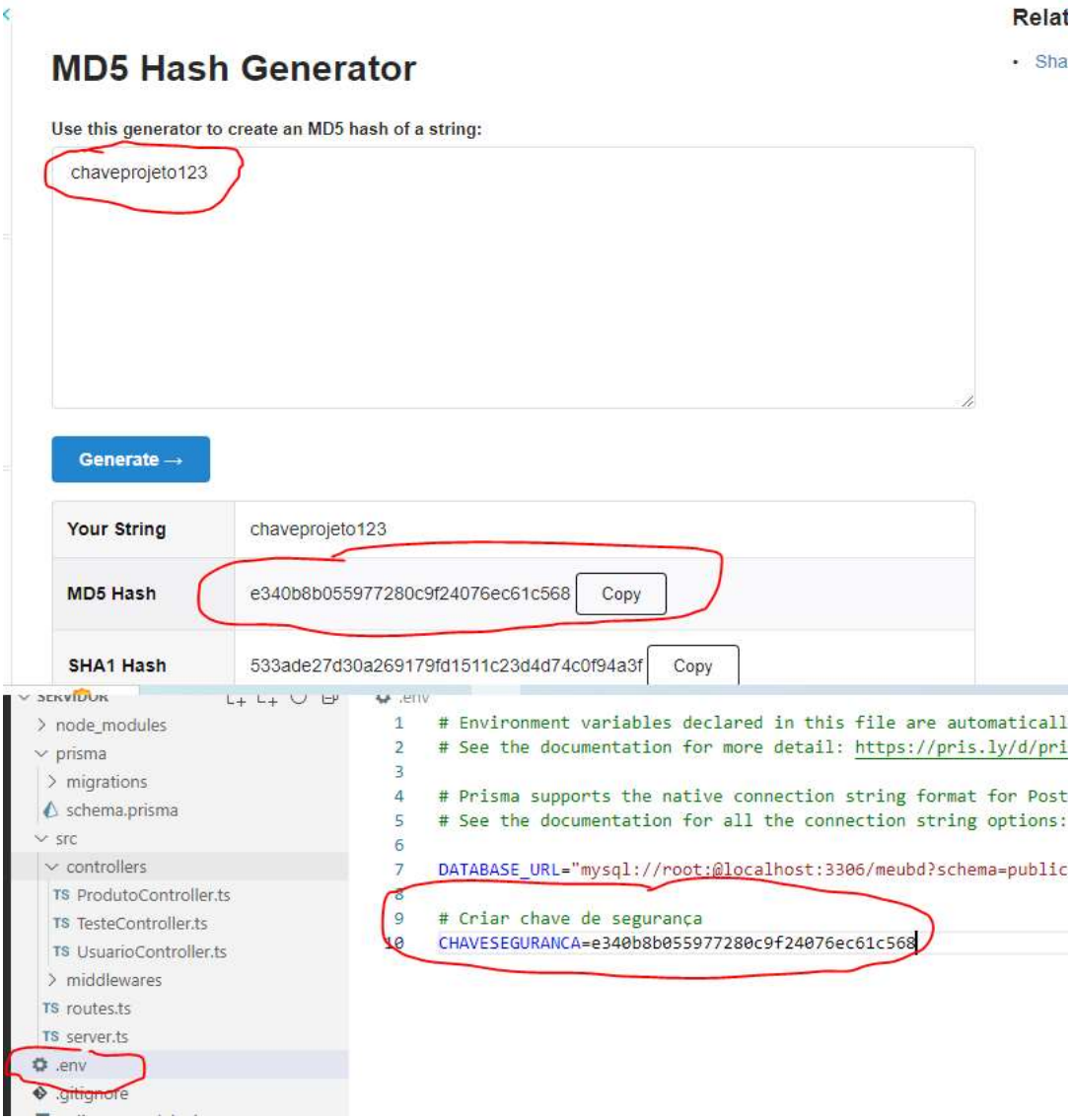
```
model Usuario{
  id Int @id @default(autoincrement())
  email String @db.VarChar(100) @unique
  senha String @db.VarChar(100)

  @@map("usuarios")
}
```

yarn prisma migrate dev --name terceira

5) Criar rotas (routes.ts) para cadastro de usuários e autenticação

```
// Cadastra usuário
Roteador.post('/usuarios', new UsuarioController().store);

// Autenticação - retorna um token de segurança se usuário for autenticado
Roteador.get('/usuarios/autenticacao', new UsuarioController().autenticacao)
```

6) Gerar chave de segurança para a aplicação e inseri-la como variável de ambiente (arquivo .env)
https://www.md5hashgenerator.com/



7) Criar UsuarioController com os método store e autenticacao

```
import {Request, Response} from 'express';
import {prisma, PrismaClient} from '@prisma/client';
import {hash, compare} from 'bcryptjs';  // pacote de criptografia
import { Secret, sign } from 'jsonwebtoken'; // sign é usado para gerar o token

class UsuarioController {
    async store(req:Request,res:Response){
        const prisma = new PrismaClient();
        const {email, senha} = req.body;
        if (email==null || senha==null) {
            return res.status(400).json({status: 'email e senha devem ser fornecidos.'});
        }
        try {
```

```
            const novoUsuario = await prisma.usuario.create(
                {
                    data: {
                        email: email,
                        senha: await hash(senha,8), // ciptografa a senha, 8 é o salto
                    },
                    select: {
                        email: true
                    }
                }
            );
            res.status(200).json(novoUsuario);
        }
        catch (erro){
            return res.status(400).json({status: 'email deve ser único'});
        }
    }

    async autenticacao(req:Request,res:Response){
        const prisma = new PrismaClient();
        const {email, senha} = req.body;
        const consulta = await prisma.usuario.findFirst(
            {
                where: {
                    email: email
                }
            }
        );
        if (consulta==null){
            return res.status(401).json({status: 'não autorizado'});
        } else {
            console.log(consulta.senha);
            console.log((await hash(senha,8)).length);
            if (await compare(senha,consulta.senha)) { // senha batem
                // gerar token
                const token = sign(
                    {
                        email: consulta.email
                    },
                    process.env.CHAVESEGURANCA as Secret,
                    {
                        subject: consulta.id.toString(),
                        expiresIn: '1d'
                    }
                );
                return res.status(200).json({token:token});
            } else {
                return res.status(401).json({status: 'não autorizado'});
            }
        }
    }
}

export default UsuarioController
```

8) Criar middleware para validação do token gerado na autenticação, usá-lo para restringir acesso ao cadastro de produtos

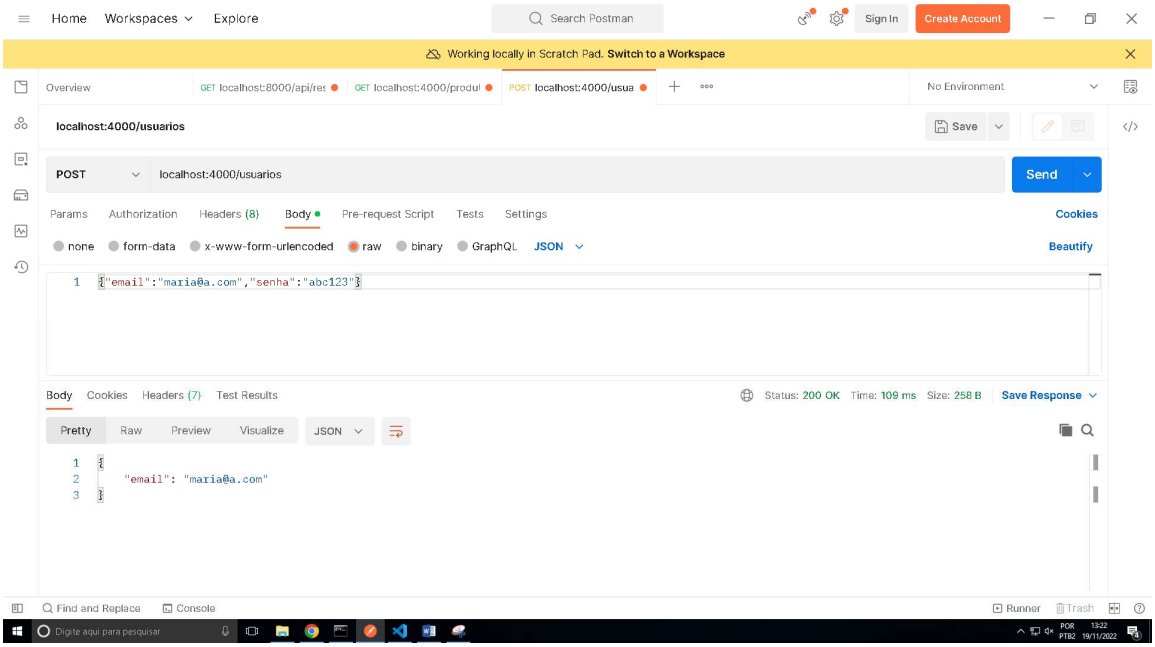

```
import {Request,Response,NextFunction} from 'express';
import {Secret, verify} from 'jsonwebtoken';
// verifica se os parâmetros da requisição são válidos
function ValidaToken (req:Request,res:Response,next:NextFunction) {
    const {token} = req.headers;

    verify(token as string, process.env.CHAVESEGURANCA as Secret, (err,decoded)=>{
        if(err) {
            return res.status(401).json({status: 'Não autorizado'});
        } else {
            console.log(decoded);
            return next();
        }
    });

}
export default ValidaToken
```

9) Incluir o middleware de validação do token nas rotas de produtos, restringindo acesso a usuários autenticados.

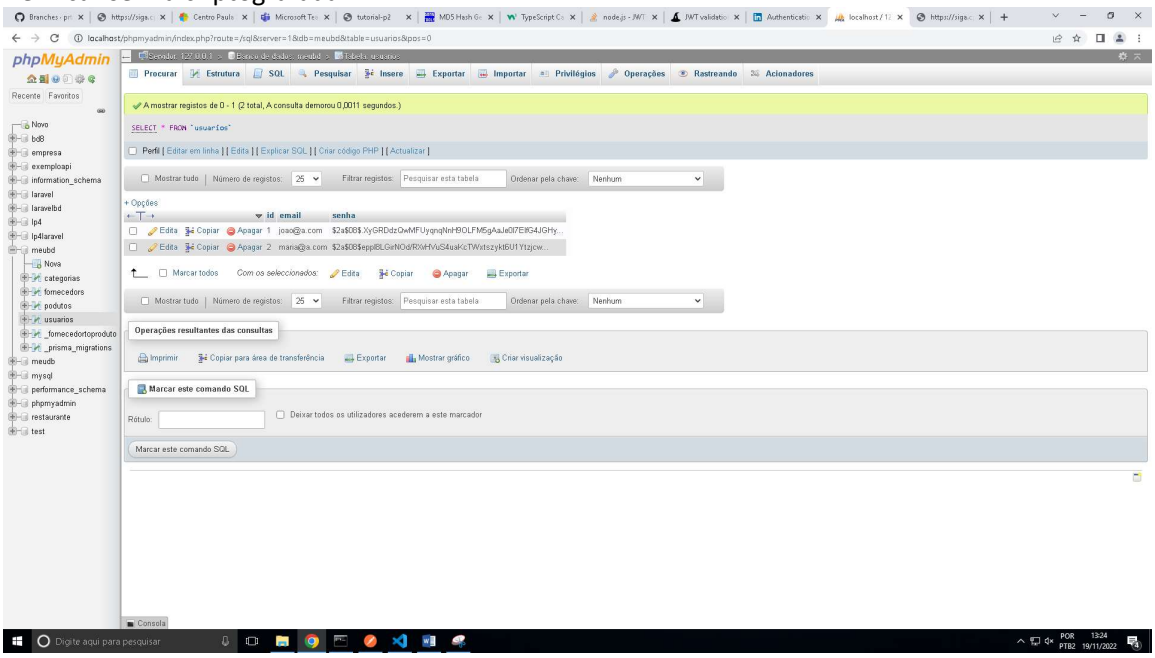
```
import ValidaToken from './middlewares/ValidaToken';

Roteador.get('/produtos', ValidaToken, new ProdutoController().index);
Roteador.get('/produtos/:id', ValidaToken, new ProdutoController().show);
Roteador.post('/produtos', ValidaToken, new ProdutoController().store);
Roteador.put('/produtos/:id', ValidaToken, new ProdutoController().update);
Roteador.delete('/produtos/:id', ValidaToken, new ProdutoController().delete);
Roteador.put('/produtos/fornecedores/:id', ValidaToken, new
ProdutoController().associarFornecedores);
```

10) Realizar testes

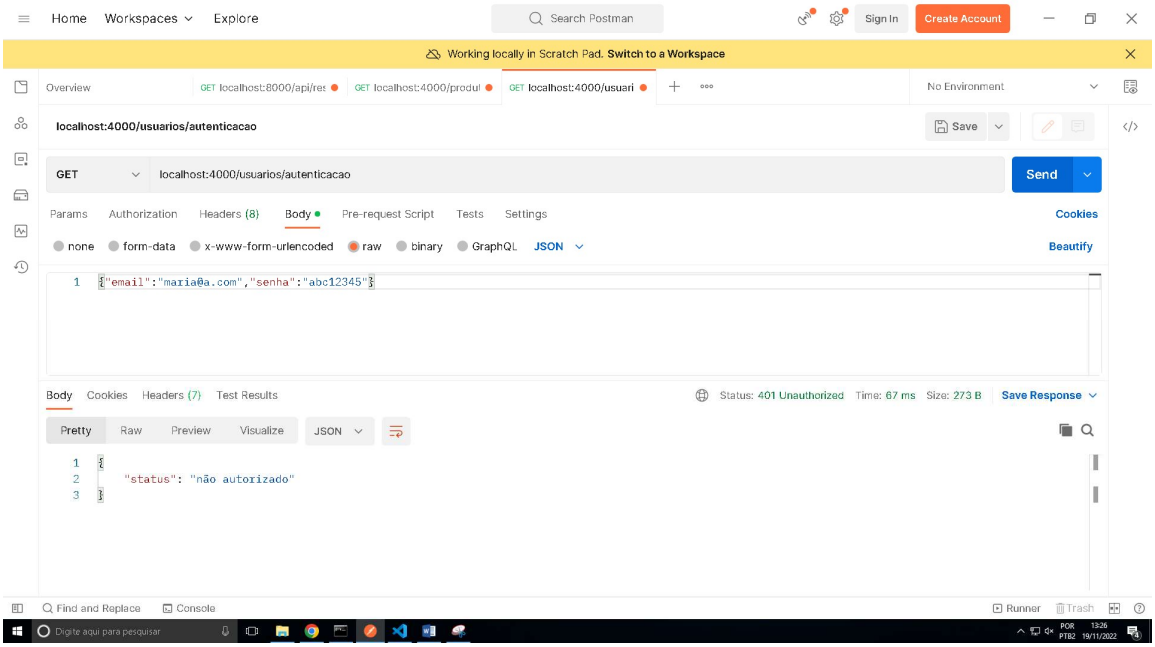
Cadastrar novo usuário



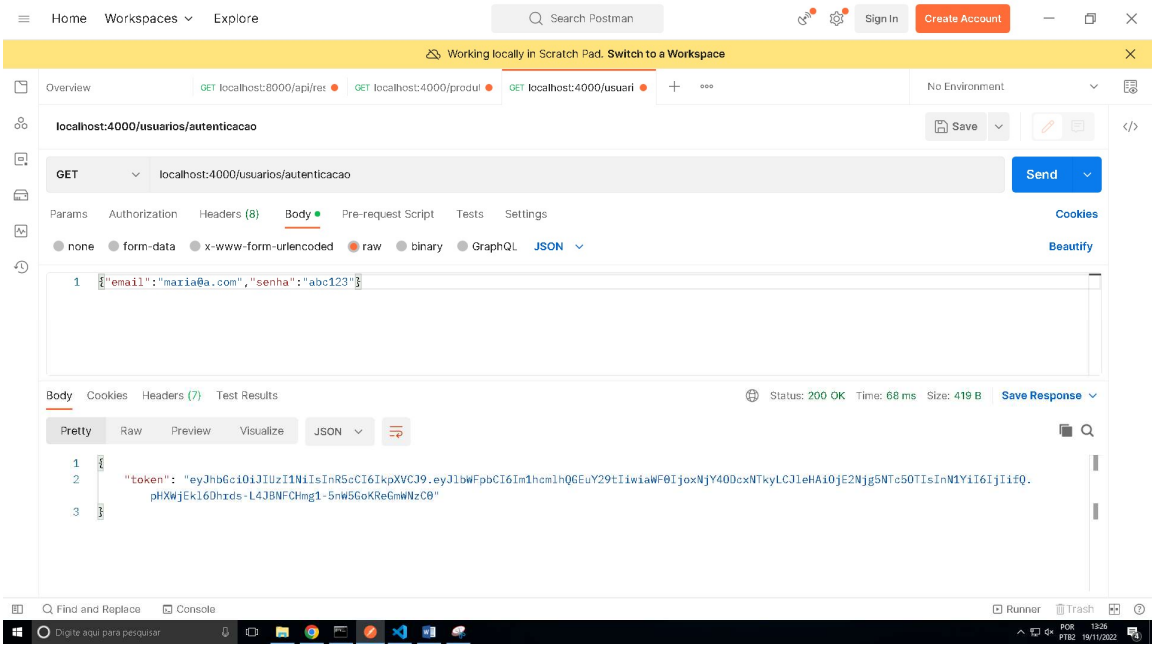
Verificar senha criptografada



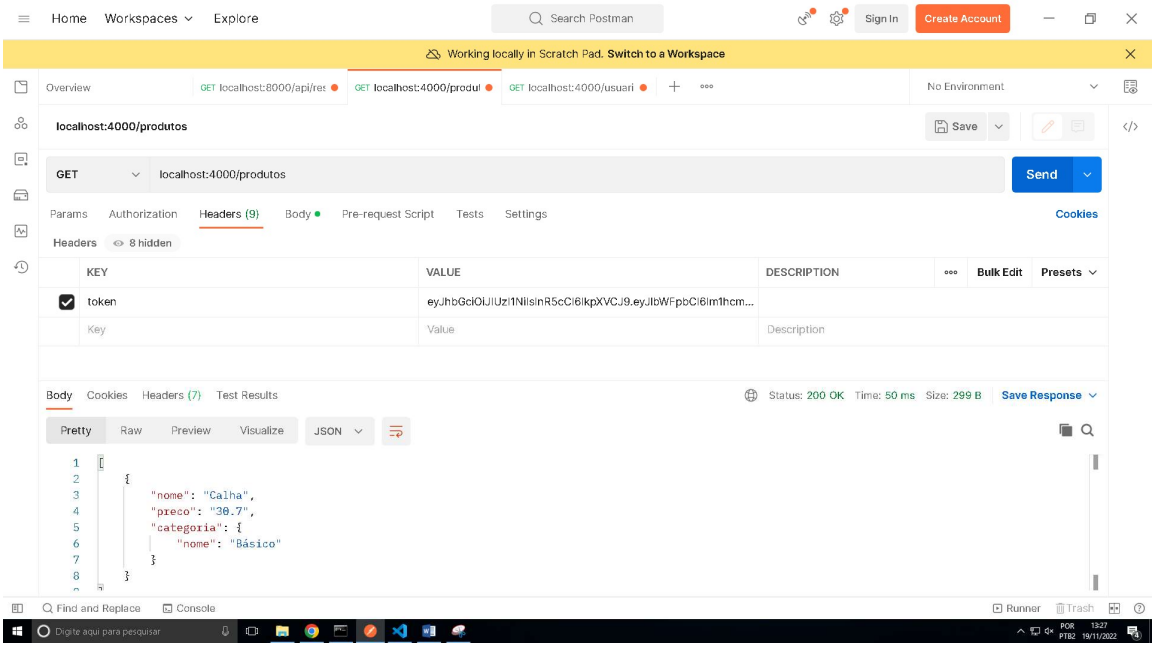
Tentar autenticar com senha inválida



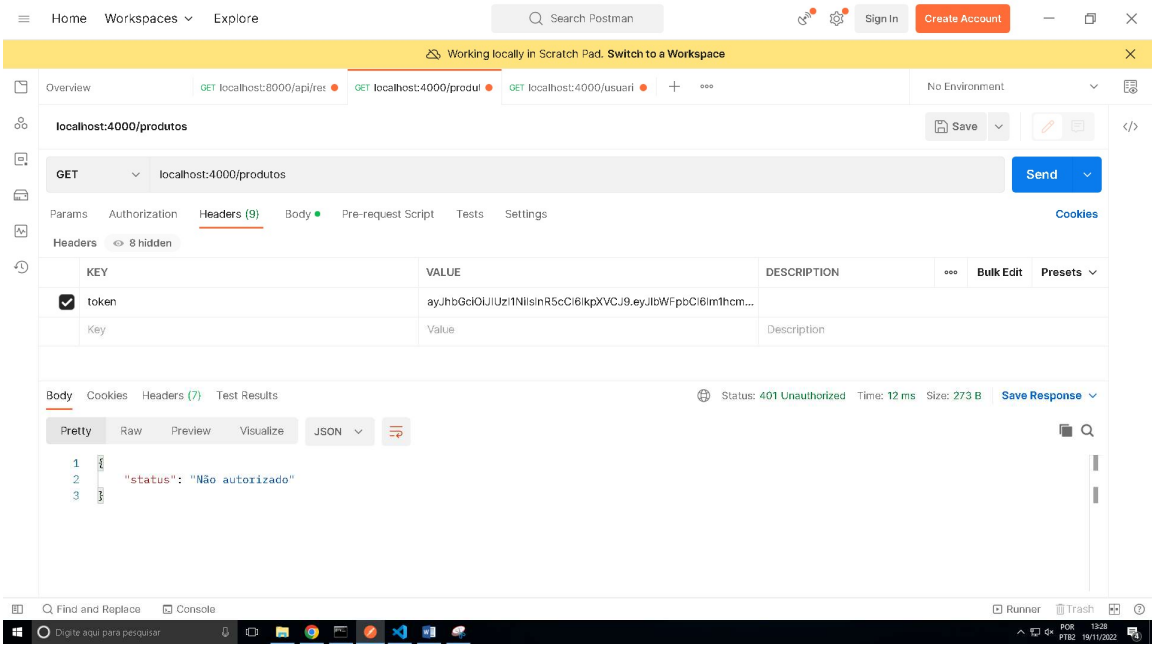
Autenticar com senha correta para obter tokem de acesso



Consultar produtos com token de acesso como header



Modificar o token (1ª letra) para forçar falha de acesso



Exercício: testar outras operações com produtos.